SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3135

Inverzna perspektivna transformacija slike ravninskog objekta

Mak Krnic

Zagreb, srpanj 2013.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Projekcijsko ravninsko preslikavanje se u području računalnog vida koristi za mnoge primjene. Neke od primjena su ispravljanje izobličenja nastalih zbog perspektivne slike na kojoj su promijenjeni odnosi linija, zrcaljenje slike, "dodavanje" perspektive na sliku ili rotacija slike.

Cilj ovog rada bio je implementirati algoritam za izračunavanje matrice inverzne perspektivne transformacije (H) na temelju korisničkog odabira četiri točke te pomoću te matrice izračunati inverznu perspektivnu transformaciju ulazne slike. Kao ulazna datoteka može se iskoristiti slika u boji (RGB[1]), bilo kojih dimenzija.

Implementacija je izvedena u jeziku C++ koristeći biblioteku OpenCV nakon što je napravljen prototip u pythonu. Nakon pokretanja od korisnika se traži da odabere četiri točke, te započinje računanje transformacijske matrice i same perspektivne transformacije slike.

Svojstva i algoritam izračuna transformacijske matrice H i perspektivne transformacije opisani su u poglavlju 2, Nakon toga u poglavlju 3 slijedi programska implementacija s odsječcima kôda u Pythonu i C++-u uz pojašnjenja te popis i informacije o korištenoj programskoj podršci i bibliotekama u poglavlju 4. U poglavlju 5 naveden je primjer izvođenja te postignuti rezultati s usporedbom brzine izvođenja programa u C++-u i Pythonu. Naposlijetku je izveden zaključak u poglavlju 6, navedena korištena literatura te sažetak najbitnijih točaka.

[1] Red Green Blue

2. Projekcijsko ravninsko preslikavanje

2.1. Prikaz točaka u ravnini homogenom notacijom

Svaka točka u ravnini u pravokutnom koordinatnom sustavu jednoznačno je određena uređenim parom koordinata (x, y) te je stoga uobičajeno ravninu poistovjetiti s $\mathbb{R}^2$. Zbog toga se točku u ravnini može prikazati vektorom $\mathbf{x} = (x, y)^\top$.

Kao što se točka u ravnini može jednoznačno predstaviti parom koordinata $(x,\ y)$, tako se pravac može prikazati jednadžbom $ax + by + c = 0$, gdje se različiti pravci dobivaju mijenjajući parametre a, b i c. Zbog tog se svojstva pravci mogu bez promjene mogu prikazati u homogenoj notaciji kao stupčani vektor $(a, b, c)^\top$. Ovaj prikaz, doduše, nije jednoznačan, budući da vektori (a, b, c) i (ka, kb, kc) predstavljaju isti pravac za svaki $k \neq 0$, ali za svaki pravac zapisan u homogenoj notaciji postoji točno jedan pravac zapisan u ortogonalnoj notaciji [5].

Da bi se točka $\mathbf{x} = (x, y)^\top$ zapisala pomoću homogenih koordinata, potrebno je uzeti u obzir slijedeće:

1. Ako i samo ako točka $\mathbf{x} = (x, y)^\top$ leži na pravcu $\mathbf{I} = (a, b, c)^\top$, onda vrijedi jednakost $ax + by + c = 0$.

2. Tvrdnja iz točke 1 se može zapisati kao skalarni produkt vektora koji prikazuje točku i homogenog prikaza pravca: $(x, y, 1) \cdot (a, b, c) = (x, y, 1) \cdot \mathbf{I} = 0$.

3. Ako i samo ako vrijedi $(x, y, 1) \cdot \mathbf{I} = 0$, onda vrijedi i $(kx, ky, k) \cdot \mathbf{I} = 0$ za bilo koju konstantu $k \neq 0$.

Iz gorenavedenoga vidi se da će točki $\mathbf{x} = (x_1, x_2, x_3)^\top$ iz projekcijske ravnine $\mathbb{P}^2$ odgovarati točka $(\frac{x_1}{x_3}, \frac{x_2}{x_3})^\top$ u euklidskoj ravnini $\mathbb{R}^2$, gdje je x_3 homogena koordinata točke $\mathbf{x}$.

Bitno je također napomenuti da se u slučaju kada je homogena koordinata točke $\mathbf{x}$ jednaka 0 kaže da je ta točka u euklidskoj ravnini u beskonačnosti.

Projekcijski prostor $\mathbb{P}^2$ opisuje se kao skup zraka u prostoru $\mathbb{R}^3$. Ako se kroz ishodište prostora $\mathbb{R}^3$ provuče pravac, točke koje se nalaze na tom pravcu mogu se predstaviti vektorima $k(x_1, x_2, x_3)$ koji se u projekcijskom prostoru $\mathbb{P}^2$ preslikavaju u istu točku.

2.2. Projekcijsko preslikavanje

Planarna projekcijska transformacija ili *homografija* je linearna transformacija nad homogenim 3-dimenzionalnim vektorom koji predstavlja homogeni zapis točke ravnine, a koja se može prikazati nesingularnom 3×3 matricom [5]:

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{2.1}$$

ili kraće

$$\mathbf{x}' \sim H \cdot \mathbf{x} \tag{2.2}$$

Važno je još primjetiti da se projekcijska transformacija ne mijenja skaliranjem transformacijske matrice H faktorom različitim od nule, pa se zbog toga matrica H naziva *homogenom matricom* jer je bitan samo omjer elemenata, ali ne i same njihove vrijednosti. Budući da postoji 8 međusobno nezavisnih omjera, kažemo da projekcijska transformacija ima 8 stupnjeva slobode.

2.3. Interpolacija

Interpolacija je matematička metoda kojom se na tememlju poznatog diskretnog skupa funkcijskih vrijednosti konstruiraju nove funkcijske vrijednosti. Diskretni se skup vrijednosti dobiva eksperimentalno ili uzorkovanjem te se interpolacija koristi kao jedna od metoda za rekonstruiranje funkcije. [10]

2.3.1. Linearna interpolacija

Linearna interpolacija jedna je od najjednostavnijih metoda interpolacije namjenjen interpolaciji točaka sa samo jednom prostornom dimenzijom, odnosno za procjenu vrijednosti funkcije koja ovisi samo o jednom parametru.

Pozunajući točke $A(x_1, y_1)$ i $B(x_2, y_2)$, linearni interpolant je dužina koja spaja te dvije točke. Pravac na kojem ta dužina leži izračunava se prema formuli za pravac kroz dvije točke [11]

$$y = y_1 + (y_2 - y_1)\frac{x - x_1}{x_2 - x_1}, y_n = f(x_n) \tag{2.3}$$

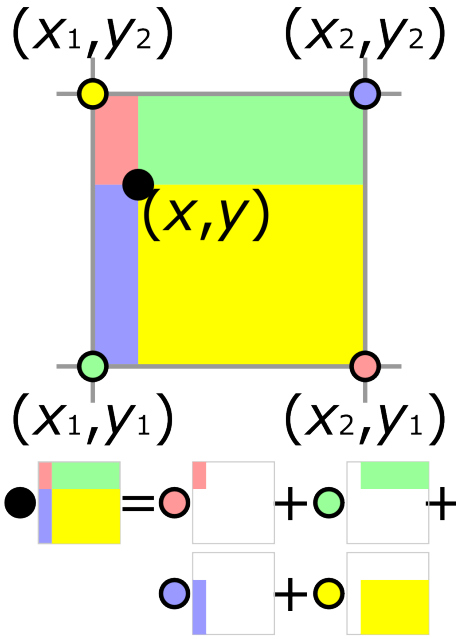
Linearna interpolacija nad skupom točaka dobiva se konkatenacijom pojedinih linearnih interpolacija između svakog para točaka, te je tada rezultat linearne interpolacije neprekinuta krivulja čija derivacija ima prekide u poznatim točkama [4] [11].

2.3.2. Bilinearna interpolacija

Bilinearna interpolacija je proširenje linearne interpolacije 2.3.1, a koristi se za interpolaciju funkcije dvije varijable na pravokutnoj 2D mreži. Provodi se tako da se provede **linearna** interpolacija prvo u jednom smjeru, a zatim u drugom [9]. Nije bitno koji je smjer prvi.

Ako su nam poznate vrijednosti funkcije u točkama $Q_{11}(x_1, y_1)$, $Q_{12}(x_1, y_2)$, $Q_{21}(x_2, y_1)$ i $Q_{22}(x_2, y_2)$, onda bilinearnom interpolacijom možemo naći nepoznatu vrijednost funkcije f u točki $P(x, y)$ koja se nalazi unutar pravokutnika omeđenog točkama Q_{11}, Q_{12}, Q_{21} i Q_{22}.

Na slici 2.1 se može vidjeti grafički prikaz izračuna vrijednosti interpolirane točke.



Slika 2.1: Grafički prikaz bilinearne interpolacije

Za primjer, pokazat ćemo algoritam kada se linearna interpolacija radi prvo po

x-osi, a zatim po y-osi.

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \,, \tag{2.4}$$

gdje je $R_1 = (x, y_1)$.

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \,, \tag{2.5}$$

gdje je $R_2 = (x, y_2)$.

Zatim nastavljamo provodeći interpolaciju po y-smjeru:

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2). \tag{2.6}$$

U ovom se radu bilinearna interpolacija koristi za određivanje boje slikovnog elementa (engl. *pixel*) u slučaju kada originalne koordinate transformirane točke nisu cjelobrojne, te se zbog toga može koristiti pojednostavljena verzija algoritma u kojoj se pretpostave koordinate točaka $(0, 0)$, $(0, 1)$, $(1, 0)$ te $(1, 1)$:

$$f(x, y) \approx f(0,0)(1-x)(1-y) + f(1,0)x(1-y) + f(0,1)(1-x)y + f(1,1)xy \tag{2.7}$$

2.4. Određivanje transformacijske matrice

Ako su nam poznate koordinate četiriju točaka na originalnoj slici I_o te znamo u koje se one točke preslikavaju na transformiranoj slici I_t, može se riješiti jednadžba (2.2) i na taj način dobiti transformacijsku matricu H [6].

S obzirom na to da nam činjenica da se transformacijska matrica H može množiti koeficijentom uvelike komplicira situaciju, kako bi pojednostavili jednadžbu (2.2) za svaku točku, možemo je zapisati na slijedeći način:

$$q_t = \lambda \cdot H \cdot q_o, \lambda \in \mathbb{R} \tag{2.8}$$

gdje je q_t transformirana točka, a q_o originalna.

Nadalje, iz jednadžbe (2.8) se može iščitati da su q_t i $H \cdot q_o$ paralelni, što znači da je njihov vektorski produkt jednak $\mathbf{0}$ [8]:

$$q_t \times (H \cdot q_o) = \vec{0} \tag{2.9}$$

Radi jednostavnijeg zapisa jednadžbi (2.1) i (2.2) možemo reći da retke matrice H prikazujemo retčanim vektorima h_i.:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} \mathbf{h_1} \\ \mathbf{h_2} \\ \mathbf{h_3} \end{bmatrix} \tag{2.10}$$

Tada dalje (2.9) postaje

$$\mathbf{q_t} \times \begin{bmatrix} \mathbf{h_1} \cdot \mathbf{q_o} \\ \mathbf{h_2} \cdot \mathbf{q_o} \\ \mathbf{h_3} \cdot \mathbf{q_o} \end{bmatrix} = \begin{bmatrix} q_{tx} \\ q_{ty} \\ q_{tz} \end{bmatrix} \times \begin{bmatrix} \mathbf{h_1} \cdot \mathbf{q_o} \\ \mathbf{h_2} \cdot \mathbf{q_o} \\ \mathbf{h_3} \cdot \mathbf{q_o} \end{bmatrix} = \mathbf{0} \tag{2.11}$$

Budući da su i transformirana (q_t) i originalna (q_o) točka zadane u ortogonalnoj notaciji, njihove su homogene koordinate q_{tz} i q_{oz} jednake 1 te (2.11) postaje:

$$\begin{bmatrix} q_{tx} \\ q_{ty} \\ 1 \end{bmatrix} \times \begin{bmatrix} \mathbf{h_1} \cdot \mathbf{q_o} \\ \mathbf{h_2} \cdot \mathbf{q_o} \\ \mathbf{h_3} \cdot \mathbf{q_o} \end{bmatrix} = \begin{bmatrix} q_{ty} \cdot \mathbf{h_3} \cdot \mathbf{q_o} - \mathbf{h_2} \cdot \mathbf{q_o} \\ -q_{tx} \cdot \mathbf{h_3} \cdot \mathbf{q_o} + \mathbf{h_1} \cdot \mathbf{q_o} \\ q_{tx} \cdot \mathbf{h_2} \cdot \mathbf{q_o} + q_{ty} \cdot \mathbf{h_1} \cdot \mathbf{q_o} \end{bmatrix} = \mathbf{0} \tag{2.12}$$

Nakon što izlučimo $\mathbf{h_1}$, $\mathbf{h_2}$ i $\mathbf{h_3}$ dobivamo:

$$\begin{bmatrix} 0 & 0 & 0 & -q_{ox} & -q_{oy} & -1 & q_{ty}q_{ox} & q_{ty}q_{oy} & q_{ty} \\ q_{ox} & q_{oy} & 1 & 0 & 0 & 0 & -q_{tx}q_{ox} & -q_{tx}q_{oy} & -q_{tx} \\ -q_{ty}q_{ox} & -q_{ty}q_{oy} & -q_{ty} & q_{tx}q_{ox} & q_{tx}q_{oy} & q_{tx} & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} \tag{2.13}$$

ili kraće

$$\begin{bmatrix} \mathbf{0}_3^\top & -\mathbf{q_o}^\top & q_{ty}\mathbf{q_o}^\top \\ \mathbf{q_o}^\top & \mathbf{0}_3^\top & -q_{tx}\mathbf{q_o}^\top \\ -q_{ty}\mathbf{q_o}^\top & q_{tx}\mathbf{q_o}^\top & \mathbf{0}_3^\top \end{bmatrix} \cdot \begin{bmatrix} \mathbf{h_1}^\top \\ \mathbf{h_2}^\top \\ \mathbf{h_3}^\top \end{bmatrix} = \mathbf{0}_3 \tag{2.14}$$

Za svaki par točaka dobivamo po dvije nezavisne jednadžbe, dakle, za četiri para dobivamo linearni sustav oblika:

$$A_{8\times 9} \cdot \mathbf{h}_{9\times 1} = \mathbf{0}_{8\times 1} \tag{2.15}$$

Uočavamo da sustav iz (2.15) ima 8 jednadžbi i 9 nepoznanica. Taj je sustav moguće rješiti zahvaljujući njegovom svojstvu homogenosti[1] metodom singularne dekompozicije (engl. *SVD, singular value decomposition*).

[1] Homogen sutstav je sustav oblika $A\mathbf{x} = \mathbf{0}$

3. Programska implementacija

U ovom poglavlju opisani su i pojašnjeni algoritmi i struktura programskog koda koji se koristi za implementaciju projekcijskog ravninskog preslikavanja

3.1. Prototip u pythonu

Kako bi se na što jednostavniji način pristupilo problemu i kako bi se fokus usmjerio prvenstveno na sam algoritam, a tek onda na specifičnosti jezika, za prototipiranje je odabran jezik *Python*.

U nastavku se mogu vidjeti ključni dijelovi koda.

Kod 3.1: Metoda u Pythonu za izračunavanje transformacijske matrice

```
1  def calculateHMatrix (originalPoints, transformedPoints):
2    orig = np.array([tuple(i) for i in originalPoints]);
3
4    transformed = np.array([tuple(i) for i in
       transformedPoints]);
5
6    A = []
7    for i in range (0,4):
8      A.append([0, 0, 0, -orig[i][0], -orig[i][1], -1,
         transformed[i][1] * orig[i][0], transformed[i][1]
         * orig[i][1], transformed[i][1]])
9      A.append([orig[i][0], orig[i][1], 1, 0, 0, 0, -
         transformed[i][0] * orig[i][0], -transformed[i][0]
          * orig[i][1], -transformed[i][0]])
10
11   A = np.array(A)
12
```

```
13    _, _, vt = linalg.svd(A)
14    h = vt[-1]
15
16    H = np.matrix([
17      [h[0],h[1],h[2]],
18      [h[3],h[4],h[5]],
19      [h[6],h[7],h[8]]])
20
21    return H
```

Odsječak kôda 3.1 prikazuje python implementaciju metode za računanje transformacijske matrice `def calculateHMatrix (originalPoints, transformedPoints ):`. `originalPoints` je polje s četiri elementa koji su uređeni parovi originalnih točaka (korisnik ih odabire pokazivačem klikom na točke), a `transformedPoints` su koordinate odgovarajućih točaka na transformiranoj slici. Ovaj dio koda odgovara odjeljku 2.4, odnosno, matematičkim formulama (2.11) - (2.15).

Kod 3.2: Metoda u Pythonu za izračunavanje pozicije i boje transformiranih točaka

```
1  def transformImage ((width, height), originalImage,
       transformationMatrix, enableInterpolation = True):
2    Hinv = linalg.inv(transformationMatrix)
3    transformedImage = array(Image.new (originalImage.mode,
         (width,height)))
4    (originalWidth, originalHeight) = originalImage.size
5
6    originalArray = array(originalImage)
7    for y in range(0, height):
8      for x in range (0, width):
9        pointTransformed = np.matrix([[x], [y], [1]])
10       pointOriginal = Hinv * pointTransformed
11
12       t = [float(pointOriginal[0][0]/pointOriginal[2][0])
           ,
13         float(pointOriginal[1][0]/pointOriginal[2][0]),
14         1]
15
```

```
16        xOrig = t[0]
17        yOrig = t[1]
18
19        if (enableInterpolation
20          and (xOrig != int(xOrig) or yOrig != int(yOrig))
21          and xOrig + 1 <= originalWidth
22          and yOrig + 1 <= originalHeight):
23
24          xOrigInt = int (xOrig)
25          yOrigInt = int (yOrig)
26
27          dx = xOrig - xOrigInt
28          dy = yOrig - yOrigInt
29
30          point = (originalArray[yOrigInt][xOrigInt] * (1-
               dx)*(1-dy)
31                + originalArray[yOrigInt][xOrigInt+1]*dx
                   *(1-dy)
32                + originalArray[yOrigInt+1][xOrigInt]*(1-dx
                   )*dy
33                + originalArray[yOrigInt+1][xOrigInt+1]*dx*
                   dy)
34
35          transformedImage[y][x] = point
36
37        else:
38          transformedImage[y][x] = originalArray[int(yOrig)
               ][int(xOrig)]
39
40    return transformedImage
```

Odsječak koda 3.2 prikazuje metodu za transformiranje slike na temelju širine i visine transformirane slike, originalne slike i transformacijske matrice. Transformacijska je matrica H namijenjena za transformaciju originalne točke u transformiranu, odnosno $q_t = H \cdot q_o$, a budući da su nam poznate trenutne koordinate *transformirane* točke, potreban nam je inverz matrice, odnosno $q_o = H^{-1} \cdot q_t$. Zato u liniji 2 inverti-

ramo transformacijsku matricu. Dalje, slijedno prolazimo kroz sve slikovne elemente i primjenjujemo transformaciju na njima. Ukoliko je uključeno interpoliranje, a koordinate trenutno originalne točke nisu cjelobrojne, provjerava se je li trenutna točka rubna desna ili donja, odnosno, zadnji ili predzadnji stupac ili redak te ako nije nad njom se provodi interpoloacija (linija 19) prema matematičkoj formuli (2.7) (linija 30).

3.2. Implementacija u C++-u

Zbog sporog izvođenja programa u Pythonu, o čemu će biti više govora u pogavlju 5, kao jezik implementacije odabran je C++. U nastavku su prikazani isječci programa u C++-u.

Kod 3.3: Metoda u C++-u za izračunavanje transformacijske matrice

```
1  Mat calculateTransformationMatrix (const vector<Point2f>&
       originalPoints, const vector<Point2f>&
     transformedPoints, bool interpolation = true) {
2    Mat A (8, 9, CV_32F);
3    for (int i = 0; i < 4; i++) {
4      A.at<float>(2*i, 0) = 0;
5      A.at<float>(2*i, 1) = 0;
6      A.at<float>(2*i, 2) = 0;
7      A.at<float>(2*i, 3) = -originalPoints[i].x;
8      A.at<float>(2*i, 4) = -originalPoints[i].y;
9      A.at<float>(2*i, 5) = -1;
10     A.at<float>(2*i, 6) = transformedPoints[i].y *
          originalPoints[i].x;
11     A.at<float>(2*i, 7) = transformedPoints[i].y *
          originalPoints[i].y;
12     A.at<float>(2*i, 8) = transformedPoints[i].y;
13
14     A.at<float>(2*i+1, 0) = originalPoints[i].x;
15     A.at<float>(2*i+1, 1) = originalPoints[i].y;
16     A.at<float>(2*i+1, 2) = 1;
17     A.at<float>(2*i+1, 3) = 0;
18     A.at<float>(2*i+1, 4) = 0;
19     A.at<float>(2*i+1, 5) = 0;
```

```
20      A.at<float>(2*i+1, 6) = -transformedPoints[i].x *
            originalPoints[i].x;
21      A.at<float>(2*i+1, 7) = -transformedPoints[i].x *
            originalPoints[i].y;
22      A.at<float>(2*i+1, 8) = -transformedPoints[i].x;
23    }
24
25    SVD svd = SVD((Mat)A, SVD::FULL_UV);
26    Mat *transformationMatrix = new Mat(3, 3, CV_32F);;
27
28    for (int i = 0; i < 9; i++) {
29      cout << (svd.vt).at<float>(8, i) << "\n";
30      transformationMatrix->at<float>(i) = (svd.vt).at<
            float>(8, i);
31    }
32
33    *transformationMatrix = transformationMatrix->inv();
34    return *transformationMatrix;
35  }
```

Odsječak koda 3.3 u C++-u za izračun transformacijske matrice odgovara odsječku koda 3.1 u Pythonu.

Kod 3.4: Metoda u C++-u za izračunavanje pozicije i boje transformiranih točaka

```
1   void transformImage (const Mat& originalImage, Mat&
        transformedImage, const Mat& transformationMatrix,
        bool enableInterpolation = true) {
2
3     Mat transformationMatrixInv = transformationMatrix;
4     CvSize transformedSize = transformedImage.size();
5     CvSize originalSize = originalImage.size();
6
7     for (int y = 0; y <transformedSize.height; y++) {
8       cerr << "Progress: " << (float)((float)y/((float)
            transformedSize.height - 1) * 100) << " %\n";
9       for (int x = 0; x < transformedSize.width; x++) {
```

```
10      Mat pointTransformed (3, 1, transformationMatrixInv
           .type());
11      pointTransformed.at<float>(0)= x;
12      pointTransformed.at<float>(1)= y;
13      pointTransformed.at<float>(2)= 1;
14      Mat pointOriginal = transformationMatrixInv * (Mat)
           pointTransformed;
15
16      Point2f originalPoint (
17        pointOriginal.at<float>(0)/pointOriginal.at<float
             >(2),
18        pointOriginal.at<float>(1)/pointOriginal.at<float
             >(2)
19      );
20
21      if (enableInterpolation
22          && (originalPoint.x != (int)originalPoint.x ||
             originalPoint.y != (int)originalPoint.y)
23          && originalPoint.x - 1 >= 0 && originalPoint.x
             < originalSize.width
24          && originalPoint.y - 1 >= 0 && originalPoint.y
             < originalSize.height
25          ) {
26
27        int xOrigInt = (int)originalPoint.x;
28        int yOrigInt = (int)originalPoint.y;
29
30        float dx = originalPoint.x - xOrigInt;
31        float dy = originalPoint.y - yOrigInt;
32
33        Vec3b point = originalImage.at<Vec3b>(yOrigInt,
             xOrigInt) * (1 - dx) * (1 - dy)
34                  + originalImage.at<Vec3b>(yOrigInt,
                       xOrigInt + 1) * dx * (1 - dy)
35                  + originalImage.at<Vec3b>(yOrigInt +
                       1, xOrigInt) * (1 - dx) * dy
```

```
36                      + originalImage.at<Vec3b>(yOrigInt +
                           1, xOrigInt + 1) * dx * dy;
37
38          transformedImage.at<Vec3b>(y, x) = point;
39        }
40        else {
41          transformedImage.at<Vec3b>(y, x) = originalImage.
              at<Vec3b>(floor(originalPoint.y), floor(
              originalPoint.x));
42        }
43      }
44    }
45  }
```

Odsječak koda 3.4 u C++-u za transformaciju slike odgovara odsječku koda 3.2 u Pythonu.

4. Korištena programska podrška i biblioteke

U ovom poglavlju naveden je popis korištene programske podrške te za svaku stavku kratak opis upotrebe u okviru ovog rada.

4.1. Python

Python je interpretirani, objektno orjentirani viši programski jezik. Njegove ugrađene strukture i metode kombinirane s dinamičkim tipiziranjem i povezivanjem čine ga veoma popularnim jezikom za brzi razvoj aplikacija (engl. *RAD – Rapid Application Development*) i brzo prototipiranje, kao i za pisanje skripti ili za povezivanje već postojećih komponenti. Python ima jednostavnu sintaksu te je fokusiran prvenstveno na čitljivost koda, čime se olakšava održavanje programa. Podržava module i pakete, čime se potiče na modularno razvijanje programa i ponovno korištenje koda. Python interpreter i standardna biblioteka su dostupni u obliku izvornog koda ili binarne datoteke besplatno i smiju se slobodno distribuirati [3].

4.1.1. Python Imaging Library (PIL)

Python Imaging Library je vanjska biblioteka za Python koja se koristi za učitavanje, konverziju, spremanje i stvaranje novih slika [7]

4.1.2. pylab

PyLab je modul biblioteke Matplotlib koji služi za iscrtavanje raznih grafičkih elemenata na slikama. U okviru ovog rada koristi se metoda ginput koja vraća točke koje je korisnik odabrao pokazivačem.

4.1.3. numpy

NumPy je paket za Python koja dodaje podršku za znastvene proračune. To je biblioteka koja pruža mogućnost rada s višedimenzionalnim nizovima i izvedenim objektima (kao npr. matricama) i sadržio funkcije za brze operacije nad nizovima [1].

4.2. C++

C++ je niži (prema današnjim mjerilima) objektno orjentirani, strogo tipizirani, kompajlirani jezik. Zbog njegove bliskosti hardveru, a i zato što se prevodi u strojni kod u cjelosti prije izvođenja, veoma se brzo izvodi. Sadrži veoma opsežne standardne biblioteke i nebrojeno mnogo vanjskih biblioteka.

4.2.1. OpenCV

OpenCV (Open source computer vision library) je biblioteka otvorenog koda koja pruža mogućnost rada s računalnim vidom i strojnim učenjem. OpenCV je stvoren kako pružio zajedničku infrastukturu za rad s računalnim vidom i kako bi olakšao i ubrzao razvoj softvera koji koristi računalnu percepciju [2].

5. Izvođenje i rezultati

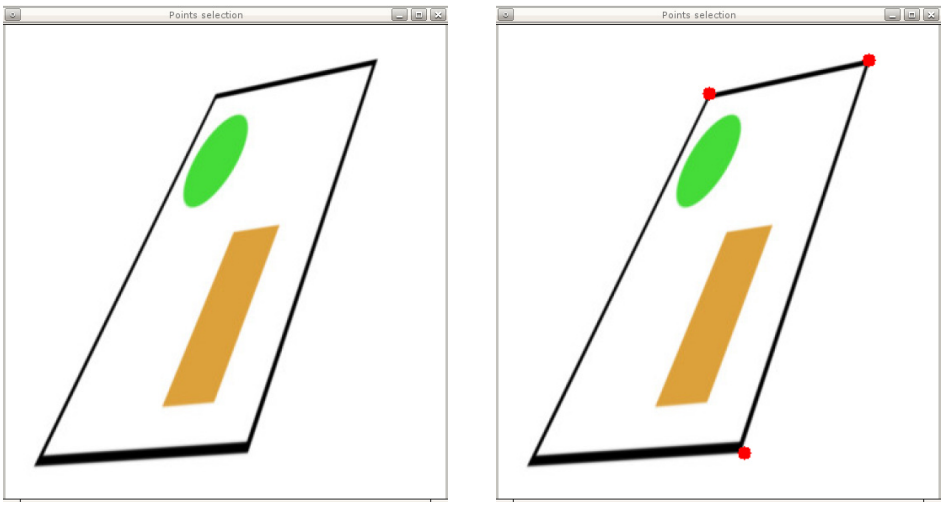
5.1. Primjer izvođenja programa

Program se pokreće iz naredbenog retka naredbom `./inverseMapping` kao u prikazu koda 5.1.

Kod 5.1: Pokretanje programa

```
1 ./inverseMapping originalna_slika.jpg
    transformirana_slika.jpg sirina_transfomirane_slike
    visina_transformirane_slike
```

Nakon toga se prikazuje prozor s originalnom slikom kao na slici 5.1a na kojemu korisnik pokazivačem odabere četiri točke koje predstavljaju kutove dijela slike nad kojim je potrebno obaviti inverznu perspektivnu transformaciju, počevši od lijevog gornjeg u smjeru kazaljke na satu (slika 5.1b).

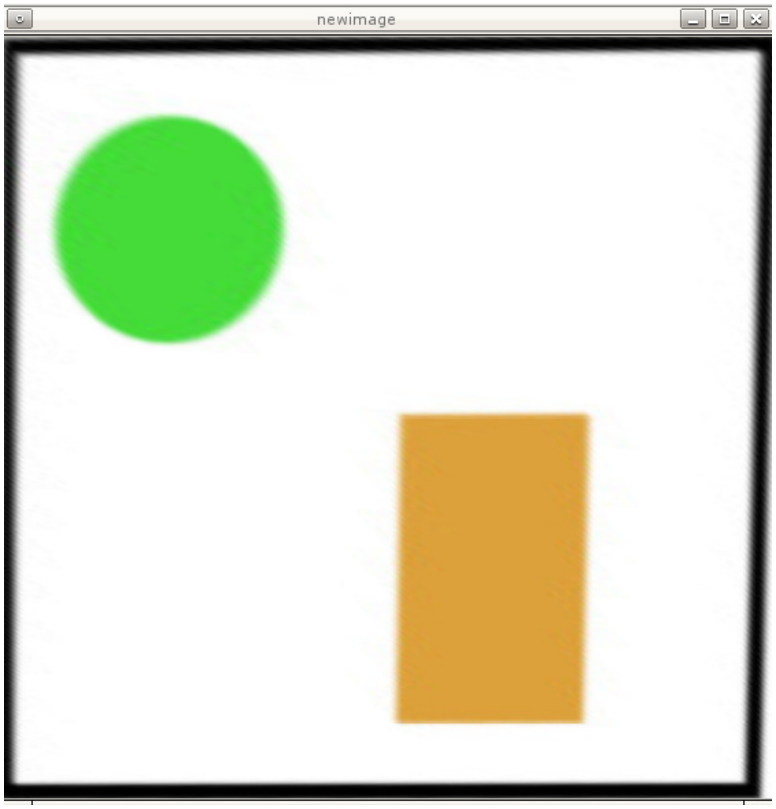


(a) Originalna slika prije odabira točaka

(b) Originalna slika za vrijeme odabira točaka

Slika 5.1: Odabir točaka

Nakon odabira točaka počinje izračunavanje transformacijske matrice, a nakon toga i sama transformacija slike čiji se napredak može pratiti u konzoli. Kada su svi slikovni elementi transformirani, transformirana se slika sprema u datoteku navedenu u naredbenom retku, a istovremeno se korisniku prikaže prozor s rezultatom, tj. istom tom transformiranom slikom (slika 5.2).



Slika 5.2: Rezultantna slika nakon transformacije

Program završava pritiskom na tipku <ESC>

5.2. Rezultati

Kao što je već bio navedeno u potpoglavlju 3 najprije je razvijen prototip u jeziku Python, a zatim, kako bi se dobile zadovoljavajuće performanse program je implementiran u C++-u. Usporedbe brzine izvođenja se mogu vidjeti u tablici 5.1.

Vrijeme izračuna same transformacijske matrice nije navedeno u tablici jer ne ovisi niti o rezoluciji slike niti o tome radi li se interpolacija ili ne te je ono približno jednako $780\mu s$ za Python program, a oko $130\mu s$ za C++ program. Mjerenja su provedena sa svim dijagnostičkim ispisima isključenima te s *niceness*[1] vrijednosti -20, kako bi se što

[1] nice je *nix program koji služi za promjenu *niceness* vrijednosti, a pomoću koje onda kernel odre-

Tablica 5.1: Brzine izvođenja programa u Pythonu i u C++-u

Jezik	Rezolucija	Interpolacija	Vrijeme transformacije (ms)
C++	100x100	NE	20
C++	100x100	DA	31
C++	1000x1000	NE	1948
C++	1000x1000	DA	3095
C++	3000x3000	DA	32541
Python	10x10	NE	43
Python	10x10	DA	45
Python	100x100	NE	3478
Python	100x100	DA	4315
Python	500x500	NE	228945

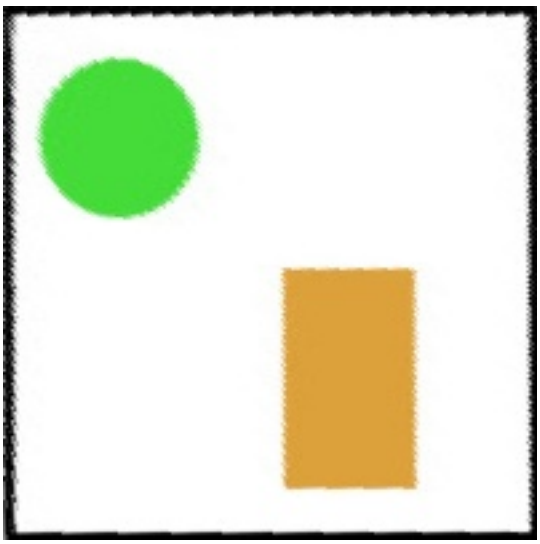
preciznije moglo izmjeriti vrijeme potrebno za samu transformaciju.

Ovi bi se rezultati mogli dodatno poboljšati raznim optimizacijama algoritma za transformaciju točaka, odnosno množenje matrica.

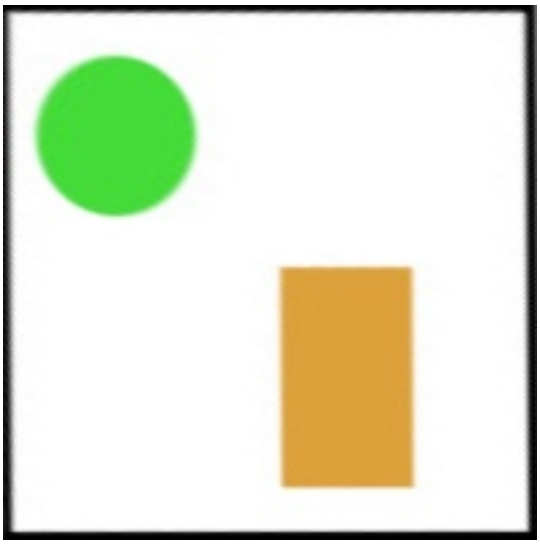
U nastavku je prikazano nekoliko primjera izlaznih slika. Ulazna slika je slika iz primjera 5.1a, originalnih dimenzija 339x369px (uključujući bjelinu). Na slici 5.3a se može vidjeti rezultat perspektivne transformacije na sliku dimenzija 200x200px bez interpolacije, a na slici 5.3b s interpolacijom.

Nedostatak u pristupu ovog programa je način odabira točaka, budući da je potrebno pokazivačem točno pogoditi kut željenog dijela slike, što se često pokaže kao veoma težak zadatak. Kako bi se korisniku olakšalo rukovanje programom, moglo bi se drugim algoritmima (npr. *Harrisov algoritam za detekciju kutova* odrediti sva potencijalna područja interesa, odnosno kutove, te bi korisnik trebao pokazivačem pogoditi samo relativnu blizinu kuta, ali ne i sam kut, nego bi taj dio posla program odradio za njega.

đuje prioritet procesa. *Niceness* može poprimiti vrijednosti od -20 do 19 ili 20, s time da veći broj označava manji prioritet. Procesi na Linuxu se uglavnom automatski pokreću s *niceness* vrijednosti 0.



(a) Bez interpolacije

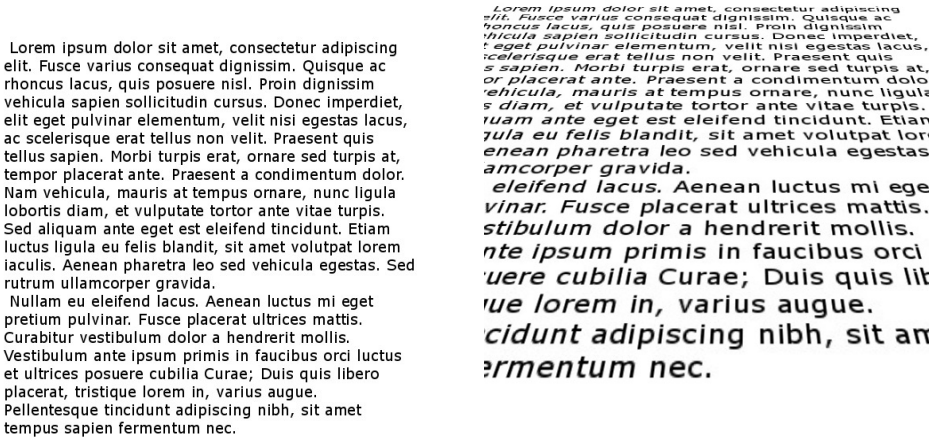


(b) S interpolacijom

Slika 5.3: Rezultat perspektivne transformacije

5.3. Druge primjene

Budući da program radi perspektivnu transformaciju, a ne nužno “ispravljanje" perspektive, moguće je “dodati" perspektivu na postojeću sliku (slika 5.4).



(a) Originalna slika

(b) Transformirana slika na koju je "dodana" perspektiva

Slika 5.4: "Dodavanje" perspektive na sliku

6. Zaključak

U okviru ovog rada ostvaren je postupak izračuna transformacijske matrice perspektivne transformacije na temelju korisničkog odabira četiriju točaka. Drugi dio rada se bavi perspektivnom transformacijom pomoću matrice dobivene u prvom koraku. Pokazalo se da za obradu slike u stvarnom vremenu (a što je potrebno u slučaju npr. ispravljanja perspektive slike snimane kamerom montiranom na vozilo za vrijeme vožnje) nije dostatan jezik Python već se zadovoljavajući rezultati postižu korištenjem C++-a. Također rezultati su pokazali da se ubrzanje od oko 25-35% može postići isključivanjem interpolacije.

Daljnji rad i istraživanje bi se moglo kretati u smjeru olakšavanja korisničkog unosa referentnih točaka, kako je već navedeno u rezultatima. Također, mogle bi se iskoristiti alternativne metode interpolacije za bolje rezultate. Dodatna ubrzanja mogla bi se postići daljnjim optimiziranjem algoritma preslikavanja točaka iz originalne u transformiranu sliku.

LITERATURA

[1] URL http://docs.scipy.org/doc/numpy/user/whatisnumpy.html.

[2] URL http://opencv.org/about.html.

[3] URL http://www.python.org/.

[4] P. Bosilj. *Optimizacija implementacije projekcijskog ravninskog preslikavanja*. 2010.

[5] R. I. Hartley i A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, 2004.

[6] Zoran Kalafatić i Siniša Šegvić. *Dinamička analiza 3D scena: geometrijski aspekti računarskog vida*.

[7] pil. Python imaging library (pil), 2013. URL http://www.pythonware.com/products/pil/.

[8] Eric W. Weisstein. Parallel vectors – from wolfram MathWorld. URL http://mathworld.wolfram.com/ParallelVectors.html. Two vectors u and v are parallel if their cross product is zero, i.e., uxv=0.

[9] Wikipedia, The Free Encyclopedia. Bilinear interpolation, Svibanj 2013. URL https://en.wikipedia.org/w/index.php?title=Bilinear_interpolation&oldid=555895667. Page Version ID: 555895667.

[10] Wikipedia, The Free Encyclopedia. Interpolation, Lipanj 2013. URL https://en.wikipedia.org/w/index.php?title=Interpolation&oldid=560095367. Page Version ID: 560095367.

[11] Wikipedia, The Free Encyclopedia. Linear interpolation, Svibanj 2013. URL https://en.wikipedia.org/w/index.php?title=Linear_interpolation&oldid=554302102. Page Version ID: 554302102.

Inverzna perspektivna transformacija slike ravninskog objekta

Sažetak

U ovom radu implementiran je algoritam za izračun matrice inverzne perspektivne transformacije na temelju korisničkog unosa četiri točke. Inverzna perspektivna transformacija je važan postupak područja računalnog vida s raznim primjenama. Za potrebe izračuna matrice proučena je matematička metoda singularne dekompozicije (engl. *SVD*), a za samu implementaciju biblioteka *OpenCV* za C++ te biblioteke *Python Imaging Library*, *PyLab* i *NumPy* za Python. Provedena su testiranja i usporedbe vremena izvođenja te je utvrđeno da se implementacija u C++-u izvodi višestruko brže od implementacije u Pythonu te da je implementacija u C++-u pogodna za obradu slike u stvarnom vremenu.

Ključne riječi: projekcijsko ravninsko preslikavanje, homografija, projekcijska transformacija, računalni vid, obrada slike, izračun matrice homografije

Inverse perspective mapping of a planar object

Abstract

This work implements the algorithm for the calculation of the inverse perspective transformation matrix based on user-selected four points. Inverse perspective mapping is an important method in the field of computer vision with various applications. For the matrix computation attention was given to the method of singular value decomposition, and for the implementation itself *OpenCV* library for C++ and the libraries *Python Imaging Library (PIL)*, *PyLab* and *NumPy* for Python. The testing was conducted noting execution times and it has been noticed that the C++ implemtation's execution time is many times shorter than that of Python's and that the C++ implementation is suitable for real-time image processing.

Keywords: projective planar mapping, homography, projective transformation, computer vision, image processing, homography matrix calculation